View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Springer - Publisher Connector

Tóth *EURASIP Journal on Audio, Speech, and Music Processing* (2015) 2015:25
DOI 10.1186/s13636-015-0068-3

EURASIP Journal on Audio, Speech, and Music Processing
a SpringerOpen Journal

RESEARCH Open Access

Phone recognition with hierarchical convolutional deep maxout networks

László Tóth

Abstract

Deep convolutional neural networks (CNNs) have recently been shown to outperform fully connected deep neural networks (DNNs) both on low-resource and on large-scale speech tasks. Experiments indicate that convolutional networks can attain a 10–15 % relative improvement in the word error rate of large vocabulary recognition tasks over fully connected deep networks. Here, we explore some refinements to CNNs that have not been pursued by other authors. First, the CNN papers published up till now used sigmoid or rectified linear (ReLU) neurons. We will experiment with the maxout activation function proposed recently, which has been shown to outperform the rectifier activation function in fully connected DNNs. We will show that the pooling operation of CNNs and the maxout function are closely related, and so the two technologies can be readily combined to build convolutional maxout networks. Second, we propose to turn the CNN into a hierarchical model. The origins of this approach go back to the era of shallow nets, where the idea of stacking two networks on each other was relatively well known. We will extend this method by fusing the two networks into one joint deep model with many hidden layers and a special structure. We will show that with the hierarchical modelling approach, we can reduce the error rate of the network on an expanded context of input. In the experiments on the Texas Instruments Massachusetts Institute of Technology (TIMIT) phone recognition task, we find that a CNN built from maxout units yields a relative phone error rate reduction of about 4.3 % over ReLU CNNs. Applying the hierarchical modelling scheme to this CNN results in a further relative phone error rate reduction of 5.5 %. Using dropout training, the lowest error rate we get on TIMIT is 16.5 %, which is currently the best result. Besides experimenting on TIMIT, we also evaluate our best models on a low-resource large vocabulary task, and we find that all the proposed modelling improvements give consistently better results for this larger database as well.

Keywords: Deep neural network; Convolutional neural network; Maxout; TIMIT

1 Introduction

In a general machine learning application, the developer just receives a large set of features, with little or no information about how the features relate to each other or even how they were obtained. In this case, a neural network expert would apply a fully connected network, which attributes no importance to the order of the features. However, the situation is quite different in the case of image recognition, where the topology of the input is clearly of crucial importance. The same holds for the spectro-temporal representation of speech signals. A speech spectrogram contains local "events"—like formant transitions and energy bursts—and then the actual position and relation between these events together define what phone we hear. Neurophysiological studies found structures in the brain that respond to local spectro-temporal modulations [1], suggesting that the approach described above might indeed by a reasonable model of speech perception.

Convolutional neural networks (CNNs) are a type of artificial neural network (ANN) that were developed for exactly those cases where the input features show local spatial correlations. Besides this, they can also handle the local translational variance of their input, which makes the network more tolerant to slight position shifts. In fact, CNNs have been successfully used in image processing for a long time (including early efforts at modelling speech [2]), but their applicability to speech recognition had not thoroughly been explored before the current renaissance of artificial neural networks. With the invention of deep

Correspondence: tothl@inf.u-szeged.hu
MTA-SZTE Research Group on Artificial Intelligence, Tisza Lajos krt. 103., H-6720 Szeged, Hungary

Springer

© 2015 Tóth. **Open Access** This article is distributed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits unrestricted use, distribution, and reproduction in any medium, provided you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license, and indicate if changes were made.

neural nets (DNNs), we learned that deep structures are very good at handling the intricate complexity presented by the acoustic modeling task [3]. After various refinements to DNNs and their training algorithms, we can now efficiently train deep networks on the huge data sets typical in speech recognition, and now HMM/DNN systems outperform conventional HMM/GMM systems on a wide variety of large vocabulary recognition tasks [4–6]. These investigations also opened up the road for convolutional networks.

Similar to DNNs, the early attempts of applying CNNs to speech recognition used the Texas Instruments Massachusetts Institute of Technology (TIMIT) dataset [7]. In contrast to image processing, in speech recognition, the two axes of a spectro-temporal representation have different roles and should be handled differently. The earliest papers applied the convolution only along the frequency axis, arguing that small time domain shifts are automatically handled by HMMs [7–9]. The supposed benefit of frequency domain convolution is that it makes the acoustic models more robust to speaker and speaking style variations. Indeed, all the studies that experimented with frequency domain convolution found that CNNs consistently outperform fully connected DNNs on the same task [7–10]. Later studies experimented with various parameter settings, network structures, and pooling strategies, including time domain convolution [8, 11–13]. Also, the experimentation has been extended to large vocabulary recognition (LVCSR) tasks, and the latest results show that CNNs can bring a 12–14 % relative improvement in the word error rate over DNNs trained on the same LVCSR dataset [14].

Here, we will explore refinement options of CNNs that have not been pursued by other authors. First, all the abovementioned studies built the convolutional networks out of sigmoid neurons [7, 9] or rectified linear units (ReLUs) [12, 14]. However, a novel type of neural activation function called the maxout activation has been recently proposed [15]. This activation function can be regarded as a generalization of the rectifier function [16], and so far, only a few studies have attempted to apply maxout networks to speech recognition tasks. These all found that maxout nets slightly outperformed ReLU networks, in particular under low-resource conditions [17–19]. Here, we show that the pooling procedure applied in CNNs and the pooling step of the maxout function are practically the same, and hence, it is trivial to combine the two techniques and construct convolutional networks out of maxout neurons.

Furthermore, a generalization to the maxout function has recently been suggested, which we will also evaluate here [20]. In the first part of the paper, we compare the various models on the TIMIT phone recognition task, as it allows quick experimenting. We find that the convolutional maxout network always performs slightly better than its ReLU counterpart and that the p-norm generalization proposed by Zhang et al. helps reduce overfitting. By switching from ReLU to maxout units, we present a relative phone error rate reduction of 4.3 % on TIMIT.

The second improvement that we apply here is the hierarchical structure best described by Veselý et al. [21]. The origins of this technology go back to the era of shallow networks, which—just as DNNs—were trained on a block of consecutive input vectors. Some authors observed that the posterior estimates obtained can be "enhanced" by training yet another network—but this time on a sequence of output vectors coming from the first network [22]. Other authors refer to this approach as the "hierarchical modeling" [23–25] or the "stacked modeling" method [26]. Two trivial improvements to this approach are when the upper net downsamples the output of the lower one [24, 27] and/or when it uses the output of some bottleneck layer instead of the uppermost softmax layer [25, 26]. Veselý's proposal was to treat this hierarchical construct as one joint model, and he also explained why the compound structure can be interpreted as a deep convolutional network [21]. Here, we experiment with his approach, but we prefer the name "hierarchical modeling" in order to avoid confusion with the more widely accepted interpretation of convolution described earlier. The tests on TIMIT will show that our convolutional maxout networks can be efficiently combined with the hierarchical modeling scheme, yielding a phone error rate reduction of about 5.5 %.

Finally, we will improve the performance of our best model by applying dropout training. The dropout method was shown to improve the generalization ability of neural networks by preventing the co-adaptation of units [28]. Dropout is now routinely used in the training of DNNs for speech recognition, and some researchers have already reported that it works nicely with maxout units as well [19, 29]. We also find it to yield a significant performance gain. Our final, best model achieves a phone error rate of 16.5 % on the TIMIT core test set, which, to our knowledge, is currently the best result on TIMIT.

Although TIMIT is suitable for the quick evaluation of a new modelling idea, it is extremely small by current development standards. Thus, in the final section of the paper, we repeat the evaluation of the best models on a LVCSR task with broadcast news recordings of 28 h. Although this corpus is still small, there are many low-resource languages for which only this amount of data is available. The evaluation on this LVCSR task shows that—albeit with smaller gain—the proposed refinements improve the recognition performance on this larger database as well.

The rest of this paper is organized as follows. First, we introduce convolutional neural networks in Section 2,

and then we apply them on TIMIT to present baseline results in Section 2.1. We present and evaluate our first proposed refinement to the baseline in the form of the maxout activation in Section 3. The hierarchical modelling approach is explained and evaluated in Section 4. The best results achieved on TIMIT by using the dropout training method are presented in Section 5. Section 6 presents the experiments on the low-resource LVCSR task, and our findings are summarized in Section 7.

2 Convolutional neural networks

Figure 1 shows the structure of the convolutional neural networks applied in this study, with the circle magnifying the operation of just one convolutional neuron. The operation of these neurons differs from standard neural units in three key ways, which can be summed up by the words "locality," "weight sharing," and "pooling" [7]. Firstly, locality means that each convolutional neuron processes only a small, localized portion of the full input space. This locality makes sense only if the input space has some inherent topology and if the feature set extracted preserves this topology. Because of this requirement, CNNs are trained on a time-frequency representation instead of the classic MFCC features. Fortunately, fully connected DNNs have also been shown to perform better on spectro-temporal representations like the energy levels obtained from mel filter banks [3]. This way, we can use the same input features for both DNNs and CNNs, which will allow a direct comparison of their results. In our case, the input to the network consists of the energy levels of 40 mel filter bank channels, and locality will mean that these 40 mel channels are divided into wider frequency bands that each cover several mel channels. The optimal size and number of frequency bands will be found experimentally but, just to give an example here, our baseline system will process the input in 17-frame blocks along the time axis, and each convolutional neuron will operate on a 17×7 spectro-temporal window.

Secondly, the convolutional units are evaluated at several, slightly shifted positions. These shifted input blocks are processed using the same weights, which property is referred to as "weight sharing" (symbolized by the dotted lines on the right hand side of Fig. 1). The interpretation is that each convolutional neuron "scans" its neighborhood for the presence of some phenomenon like a formant transition. In speech recognition experiments, the shifting is normally applied only along the frequency axis to help account for formant shifts caused by speaker variations or speaking style [7]. In our implementation, the amount of shifting will be measured in mel channels. For example, a pooling size r will mean that the convolutional units process r versions of their input window shifted by $0, 1, \cdots, r-1$ mel banks. Extending the shifting to the time axis seems unnecessary, as hidden Markov models inherently handle time shifts. Recently, both Abdel-Hamid et al. and Sainath et al. experimented with convolution along time, and the improvements indeed proved negligible [11, 12].

Thirdly, the neural activations got at the various positions are turned into one value in the "pooling" step. Several strategies exist for this, the classic one being max-pooling [7], but other, more sophisticated pooling formulas have also been proposed. For example, Abdel-Hamid et al. investigated weighted softmax pooling [11], while

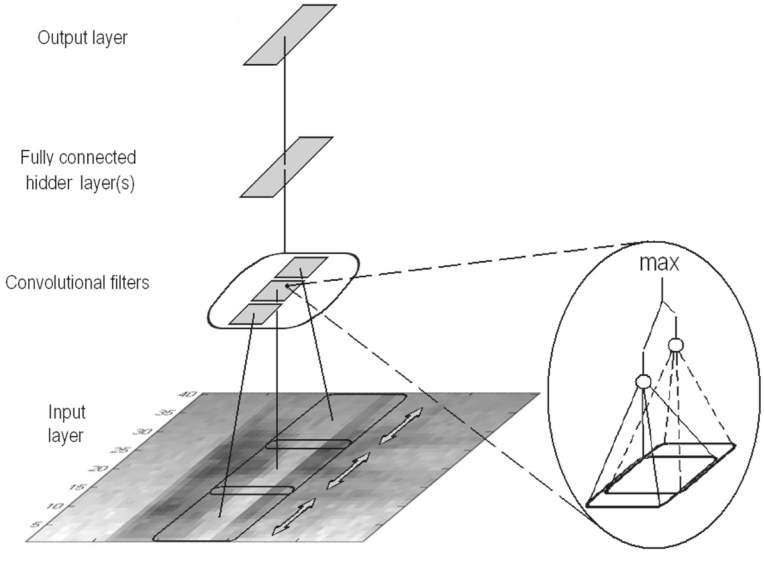

Fig. 1 A schematic diagram of the CNN network structure applied here. The *circle* on the right magnifies the operation of one convolutional neuron

Sainath et al. studied p-norm pooling and stochastic pooling [12]. However, none of these proved significantly better than simple max-pooling. Here, we will first apply max-pooling, but later, we will also experiment with p-norm pooling.

Having discussed the operation of convolutional neurons, let us describe the structure of the whole network. There are two strategies for combining the information got from the neurons assigned to different spectral bands. Abdel-Hamid et al. argue that the spectral phenomena occurring in different spectral regions are different, so each band should have a separate set of convolutional neurons. This scenario is known as the limited weight sharing (LWS) strategy [7]. In the full weight sharing scheme (FWS), all neurons are applied across all spectral regions, so the neurons encounter a more elaborate learning task. However, Sainath et al. argue that with a large enough number of hidden units, FWS can attain the same performance as LWS, while it is technically simpler and allows the stacking of convolutional layers [12]. In this study, we applied limited weight sharing, which is shown in Fig. 1 by the division of the convolutional layer into smaller blocks.

In our network, the outputs of the convolutional filters are concatenated and processed further by three additional, fully connected layers. However, we should mention here that more elaborate network structures are also possible. For example, Sainath et al. achieved the best performance by stacking two convolutional layers plus four fully connected layers [12]. The same team also experimented with combining convolutional and nonconvolutional layers within the same network [13]. Abdel-Hamid et al. improved their results by combining various pooling sizes within the same system [8].

2.1 Baseline results on TIMIT

We evaluated all the proposed algorithms via phone recognition tests on the well-known TIMIT database. We used the standard 3696 "si" and "sx" sentences as the training set, while the testing was performed on the core test set of 192 sentences. A randomly selected 10 % of the training set was held out as the development set for the neural network training process. The same set of sentences was used for tuning the meta-parameters of the various network configurations. All the experiments used a phone bigram language model estimated from the training data. While the decoder operated with 61 phone labels, during evaluation, these were mapped to the set of 39 labels proposed by Lee and Hon [30]. During decoding, no attempt was made to fine-tune the language model weight and the phone insertion penalty parameters; they were just set to 1.0 and 0.0, respectively. We made this decision in order to keep the results comparable with those of some earlier studies (e.g., [3]).

Creating context-dependent (CD) phone models is vital for getting a good performance with standard HMMs, and now, it is widely accepted that CD modelling is also beneficial for HMM/DNN hybrids trained on large vocabulary tasks [4, 5]. We found earlier that applying CD states as the network training targets is useful even for such a small corpus as TIMIT [25]. Hence, in all the experiments reported here, we used a tied state set that was obtained by training a conventional CD-HMM (using HTK). The decision tree-based state clustering tool of HTK produced 858 tied states, and evaluating the phone models in forced alignment mode yielded the training targets for each frame of speech. The tied state set we applied here is the same as that used in our earlier study [25].

As explained earlier, CNNs require a representation that preserves the time-frequency topology of the original input, so they cannot operate with MFCC features. Here, we worked with a mel-scaled time-frequency representation which was extracted using the "FBANK" feature set of the HTK toolkit. We had the opportunity to work with exactly the same features as those used in [3], as the authors kindly provided us with the corresponding HTK config file. This preprocessing method extracted the output of 40 mel-scaled filters and the overall energy, along with their Δ and $\Delta\Delta$ values, altogether yielding 123 features per frame.

The input vector to a convolutional neuron is constructed as follows. Let us assume that the input window of the neuron is 17 frames times 7 mel bands. This input is extended with the global energy of the frame as the 8th "band", which gives $17x8 = 136$ features. For all these, the corresponding Δ and $\Delta\Delta$ values are also included, altogether resulting in a feature vector of 408 features.

In each case, the neural networks were trained with standard backpropagation, with 100 data vectors per mini-batch. The weights were initialized following the scheme proposed by Glorot et al. [31], and no form of pre-training was applied. As the initial learn rate, we always used the largest possible value that gave numerically meaningful error rates. The initial learn rate was held fixed while the error rate on the development set kept decreasing. Afterwards, it was halved after each iteration, and the training was halted when the decrease in the error rate dropped below 0.1 %. We found that our maxout networks gave good results only when a large momentum value of 0.9 was used.

In order to prevent an unbounded growth of the network weights, we scaled down the weights after each epoch so that the $L1$ norm of each layer remained the same as it was after initialization. The error function we optimized was the usual frame-level cross entropy cost, though sequence-level training criteria—which are routinely used with HMM/GMMs—are now becoming more

popular with HMM/DNN hybrids as well [32, 33]. During decoding, the DNN outputs are used as state observation probability estimates of an HMM, following the scheme of hybrid HMM/ANN systems [34]. Though formally the DNN posterior estimates should be divided by the prior probability values, we omitted this division step, as this way we got consistently better phone recognition results on TIMIT. As the HMM decoder, we applied a modified version of the HVite tool of HTK.

To have a baseline result with a fully connected network, we trained a fully connected DNN with 4 hidden layers and 2000 hidden units per layer. The input to the network consisted of 17 consecutive frames of the 123 FBANK features described earlier. Apart from the softmax output layers, all hidden neurons were ReLUs [16]. In an earlier study, we showed that DNNs of rectifier units can attain the same accuracy as a sigmoid network, but without pre-training [35]. Other authors reached the same conclusions, but using much larger datasets [36–38]. Our fully connected ReLU DNN got an error rate of 20.6 % on the TIMIT core test set.

In the next step, we replaced the lowest layer of the above network with convolutional ReLU units. Here, we had to decide on the number and size of weight sharing spectral bands. The other meta-parameter we had to tune was the pooling size r. We experimented with the number of LWS bands running from 4 to 8, with the height of the filters being chosen accordingly, allowing a slight overlap between the filters. The number of neurons was always chosen so that the global number of parameters remained the same as that for the baseline fully connected DNN. Table 1 shows the error rates we obtained on the development set by varying the size of the filters. The pooling size in these experiments was set to $r = 3$. All the configurations we evaluated gave practically the same result, with no significant difference. For the subsequent experiments, we chose the 7×7 configuration, as the frame error rate was the lowest in this case. By comparison, while our filters are 7 mel channels wide, Abdel-Hamid et al. used a filter size of 8 channels [7], and Sainath et al. preferred a filter size of 9 channels [9].

Table 1 Phone error rates of the convolutional ReLU network as a function of the number and width of the frequency bands

Number of LWS bands	Width of filters	Number of units per band	Error on development set (%)
4	12	768	16.6
5	10	638	16.6
6	8	554	16.9
7	7	485	16.6
8	6	433	16.5

Next, we varied the pooling size r between 1 and 6. As shown in Fig. 2, a pooling size of 5 gave the best result on the development set, though the scores on the test set do not seem to be significantly different for any r values between 2 and 6. In comparison, Abdel-Hamid found $r = 6$ to be optimal for TIMIT [7], while Sainath et al. reported that $r = 3$ performed best on other databases [9]. We think that the optimal value may depend both on the database used and the filter size. We note that it also makes sense to combine various pooling sizes within the same model [8]. A very interesting further observation is the good performance of pooling size $r = 1$. In this configuration, no shifting and pooling occurs, so its large gain simply comes from dividing and processing the input in smaller frequency bands.

Lastly, we attempted to vary the size of the filters along the time axis. As regards fully connected DNNs, a similar experiment was performed by Mohamed et al. They found that increasing the context size from 9 to 17 or even 27 frames improves the results, but beyond this, the error rate starts to rise [3]. One might expect that CNNs can perhaps handle longer time contexts, as their input is divided into frequency bands, so the number of inputs per neuron is fewer. However, as Fig. 3 shows, we got very similar results. While there is a small improvement by changing from 9 to 17 frames, the scores saturate at 33 frames and even start to deteriorate at 49 frames. Thus, we decided to use 17 frames in the subsequent experiments. By comparison, Abdel-Hamid et al. used 15 frames [7], while Sainath et al. preferred to work with just 9 frames [14].

In summary, in this section, we built a deep CNN out of ReLU units, and we found that the best convolutional model reduced the error on the core test set by almost 2 % absolute (9 % relative), compared to a fully connected DNN. In Section 3, we propose several modifications to the convolutional model, improving its performance even further.

2.2 Comparing the speaker invariance of DNNs and CNNs

The position of formants for the same phone may vary slightly between different speakers [7] or with different speaking styles [2]. The pooling operation theoretically makes the CNNs more tolerant to these small shifts in formant frequencies. Consequently, CNNs should have a smaller performance drop for test speakers whose voice is different from that of the training speakers. To validate this assumption empirically, we examined the per-speaker scatter of the DNN and CNN recognition scores. Fortunately, TIMIT allows such an evaluation, as it contains samples taken from 630 speakers, and all sentences are speaker-annotated. We calculated the recognition accuracies for each 24 speakers of the test set, for the baseline fully connected DNN, and for the CNN with pooling size

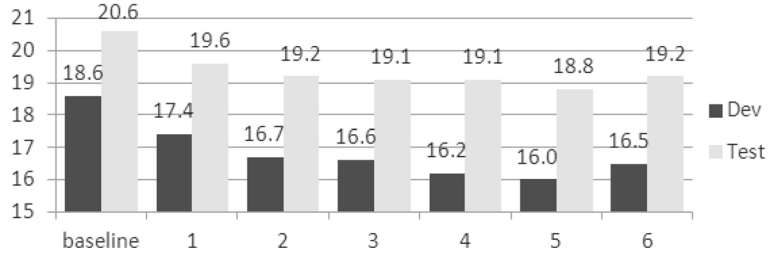

Fig. 2 Phone error rate as a function of the pooling size. The baseline score was obtained by using a fully connected DNN

$r = 5$. Analyzing the results, we found that switching from the DNN to the CNN not only increased the mean of the per-speaker recognition accuracies but also decreased their variance by about 5.7 %. This justifies our assumption that the CNN is more successful in handling speakers not seen during training.

Clearly, several other factors also contribute to the superiority of the CNN. Looking at Fig. 2, we see that the CNN already outperforms the DNN with a pooling parameter of $r = 1$. In this case, no actual shifting and pooling occurs, so the increased shift tolerance cannot account for the performance gain. CNNs were recently shown to outperform DNNs under channel-mismatched conditions [39], which may be the source of improvement at $r = 1$. As, unfortunately, TIMIT has no annotation of the recording conditions, which would allow us to test this hypothesis empirically. Lastly, we mention that Huang et al. also found CNNs to be more robust against background noise and to perform better in distant speech recognition [39].

3 Maxout neural networks

For decades, the sigmoid function was considered to be the ideal activation function for the hidden neurons of neural networks. However, with the invention of DNNs, the quest for alternative activation functions has also revived. A good example is the success of the rectifier function, which seems to be a better choice than sigmoid when building deep nets, and it is growing more popular in the speech community [35–38]. Recently, Goodfellow et al. suggested a generalization of the rectifier function where the maximum is taken over the linear activation of several neurons [15]. Stated formally, let us define the output of a neuron as

$$o = \phi(z), \quad z = \vec{w} \cdot \vec{x} + b\,.$$

That is, first, the linear activation z of the neuron is calculated from the input vector $\vec{x}$, the weight vector $\vec{w}$, and the bias term b, and then z gets transformed by the nonlinear activation function ϕ. Conventionally, the sigmoid function is used as ϕ, while the rectifier function is defined as $\max(z, 0)$ [16]. The maxout function proposed by Goodfellow et al. divides the N neurons of a given layer into L groups of size K ($N = K \cdot L$) and the output of the lth group is calculated as

$$o_l = \max_{k=0}^{K-1} z_{lK+k}, \quad l = 0, ..., L-1\,.$$

When compared with the rectifier function, we see that the rectifier activation is a piecewise linear function consisting of two pieces, with one of them being fixed. The maxout function extends this to K pieces, all of them being parametric. This increased flexibility was shown to result in an increased performance on image recognition tasks [15].

Several studies have already investigated the performance of deep maxout networks in speech recognition. All these studies found that maxout networks perform better or at least no worse than ReLU networks, and the biggest gains were reported under low-resource conditions [17–19]. Although some of these studies involved experiments with CNNs as well, these applied the maxout activation only in the fully connected layers [40]. To our knowledge, the only exceptions where the maxout activation was extended to the convolutional neurons as well are the studies of Cai et al. [41] and Renals and Swietojanski [42]. Below, we will explain how convolutional and maxout neurons are related, and we will present our solution for the swift evaluation of convolutional maxout units.

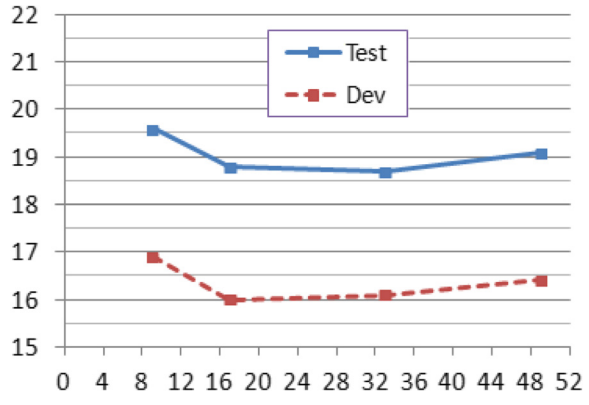

Fig. 3 Phone error rate as a function of the input context size. The context size is given in terms of frames

3.1 Applying the maxout activation to convolutional units

The maximization performed by the maxout function is technically the same as the max-pooling step applied in convolutional networks. In CNNs, the pooling is performed over neurons that process *different input* vectors using the *same weights*. In maxout networks, it is applied over *different neurons* that process *the same input*. However, as depicted on the left hand side of Fig. 4, the result in both cases is just a set of neural activations that have to be pooled, and the pooling operator does not need to know how the actual values were obtained. Hence, exploiting the associativity of the max operator, we can perform the two types of pooling operations in one go, as illustrated on the right hand side of Fig. 4. This is how we implemented the convolutional maxout units of our network, and in the following, we shall investigate how this technology works in practice.

3.2 Improvements to maxout

Several authors reported that max-pooling CNNs are inclined to overfit the training data [43]. A similar behavior was observed with maxout networks as well [29]. In both cases, the reason is that during training, the max function propagates the gradient only to the unit that gave the largest activation, so the remaining units do not get updated. As a remedy, Zeiler et al. proposed to apply stochastic pooling, whose method chooses its output (and hence the backpropagation path) randomly, with a probability proportional to the value of the corresponding activation [43]. While they obtained good results with stochastic pooling CNNs on image recognition tasks, the experiments by Sainath et al. did not justify its usefulness in speech recognition. They conjectured that perhaps the much larger amount of data typical in speech recognition already alleviates the problem of overfitting, so that the benefits of stochastic pooling are not observed [12]. Meanwhile, Cai et al. also evaluated stochastic pooling, but within the framework of maxout networks. In their tests, stochastic pooling significantly outperformed standard max pooling in all cases [29]. These contradictory results suggest that the stochastic pooling technique will require more evaluations in the speech domain. Zhang et al. experimented with applying the p-norm function instead of max pooling [20]. Using the earlier notation, p-norm defines the output of the lth group as

$$o_l = \left(\sum_{k=0}^{K-1} |z_{lK+k}|^p \right)^{1/p}.$$

Intuitively, the p-norm acts as a smoothed version of max-pooling, where the pooled units contribute to the result proportional to their absolute value. It also behaves similarly during training: all grouped units get updated, with the error being proportional to the corresponding activation value. Hence, one can expect p-norm pooling to decrease overfitting in a way that is similar to the effect of stochastic pooling. However, the experimental results on speech data are again contradictory: Sainath et al. found that p-norm pooling brought about no improvement to their CNN [12]. In the framework of maxout networks, Zhang et al. got better results with p-norm than with max pooling. They obtained the best scores with $p = 2$ and with a group size of $K = 10$ [20].

In our experiments with p-norm pooling, we set p to 2, following Zhang et al, but in our first tests, the group size was set to 2, which was found to be optimal for maxout networks [17–19]. Our tests quickly revealed that our p-norm implementation faces difficulties with propagating the error back to lower layers. Zhang et al. applied discriminative pre-training (DPT) in their study, which is a layer-by-layer building strategy for deep networks [5]. We added DPT to our code, and the results improved dramatically, but the error rates we obtained still fell short, in favor of max pooling (the corresponding results are shown in rows 2 and 4 of Table 2). For comparison, we also tried DPT with maxout, and for this pooling function, DPT made no difference (see row 3 of Table 2). In general, we think that p-norm pooling requires a careful normalization of the weights/derivatives during training, for which we have not yet found the proper strategy.

Looking for a way to combine the efficient learning property of the max function with the smoothing behavior of p-norm, we came up with the following solution. During pre-training, for each presentation of each training sample, the network was randomly considered to be a maxout network of a p-norm network. This way, within a batch, the p-norm error update rule was evaluated for q percent of the input vectors, while the standard maxout rule was applied to the remaining ones. As the weights are updated following the average of the error within the batch, we hoped that this stochastic mixing of the two error functions would help the maxout network avoid local minima. The best results were obtained with $q = 0.2$, and Fig. 5 shows an example of how this "hybrid"

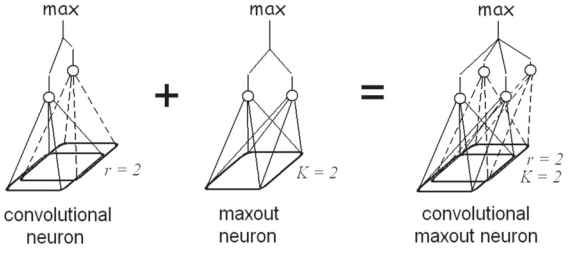

Fig. 4 The implementation of convolutional maxout neurons. The maximization over convolutional positions and maxout groups can be performed in one go

Table 2 Phone error rates of the CNN for various types of pooling functions

Network type and training method	Frame error on developmental set (%)	Phone error	
		Developmental set (%)	Core test (%)
ReLU	34.7	16.0	18.8
Maxout	34.3	15.7	18.3
Maxout, DPT	34.2	15.8	18.1
2-norm, DPT	35.5	16.3	18.9
Maxout, hybrid DPT	33.4	15.6	18.0

training method influences the error curves during training. Clearly, although the learning becomes slower and requires two more iterations, the final error rate is much lower on the train set and slightly smaller on the development set. We applied this training method only during pre-training, so after the addition of the last hidden layer, the network was trained as a normal maxout net. After this training step, as Table 2 shows, our CNN attained a frame error rate that was 0.8 % better than with the conventional training process. Unfortunately, for the actual example, this improved frame error rate is not reflected by the phone error rates. In spite of this, we applied this hybrid pre-training method in all the remaining experiments.

In the last parameter tuning experiment, we looked for the optimal group size K. Similar to the case of comparing the ReLU net with the maxout net, we took care to choose the number of units so that the global number of weights remained about the same. As maxout units have one output *per group*, this means that by increasing the group size, we can increase the number of neurons as well. As shown in Table 3, varying the group size from 2 to 5 did not change the recognition accuracy scores. This accords with the findings of other authors working with maxout [17, 19]. In all the remaining experiments, we used a group size of $K = 2$.

3.3 Comparing ReLU and maxout CNNs on TIMIT

We evaluated our maxout CNN on TIMIT, using the results got with the ReLU CNN (cf. Fig. 2) as a base of comparison. This time, we did not vary the number of the frequency bands because previously, the system seemed unaffected by this parameter value. However, as the convolutional and the maxout pooling steps are done jointly in our maxout CNN network, we repeated the experiment that looked for the optimal pooling size r. Figure 6 compares the performance of the ReLU and the maxout CNN with various pooling sizes. As can be seen, the maxout CNN outperformed the ReLU CNN for all r values, for both the development and the test set. The error rates have a similar trend for both networks and take their minimum at $r = 5$, suggesting that the joint pooling operation

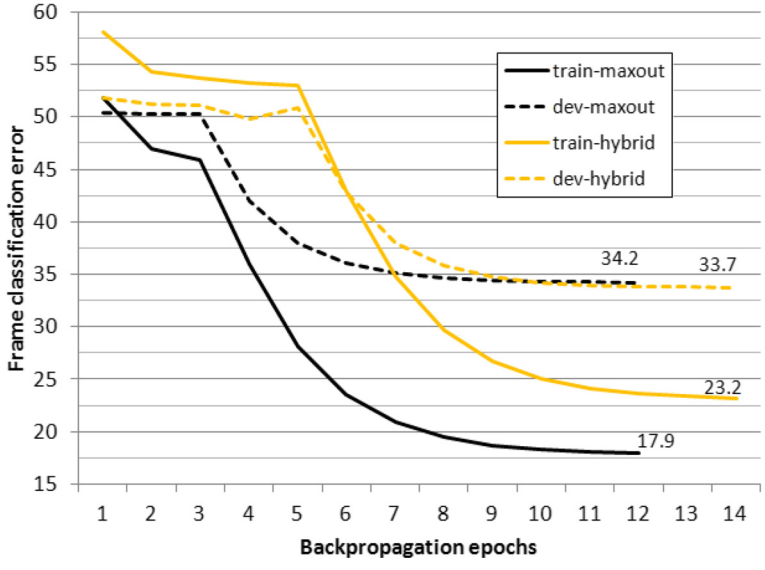

Fig. 5 The progress of phone error rates during training. The curves are shown for the maxout and the hybrid methods, for the train and the development sets

Table 3 The effect of the group size on the performance of the maxout CNN

Group size	Layer size		Development error (%)	Test error (%)
	Full	Convolutional		
2	2714	756	15.6	18.0
3	3204	960	15.5	18.0
4	3584	1160	16.0	18.1
5	3890	1340	16.0	18.2

works alright. Compared to its ReLU counterpart, the maxout CNN at $r = 5$ attained a 2.5 % relative error rate reduction on the development set and 4.3 % on the test set.

4 Hierarchical modelling

In the so-called tandem model, the ANN outputs are used as features of a conventional HMM, virtually stacking a GMM on top of the ANN [44]. Perhaps the success of this model motivated the idea of stacking two neural networks on each other. In this hierarchical model, both ANNs use several frames of input context—corresponding to a context of acoustic features for the lower network and a context of state posterior estimates for the upper one. Ketabdar et al. observed that with this method, the ANN posterior estimates can be significantly "enhanced" [22]. Pinto et al. gave a detailed analysis of how this hierarchical model exploits the information in the temporal trajectories of the posterior feature space [23]. In conventional HMM/ANN training, the number of ANN outputs is the same as the number of HMM states. Concatenating several of these output vectors may result in a feature set that is prohibitively large for training the second network of the hierarchy. This can be overcome by discarding the uppermost layer(s) of the first network and using the activation values of some hidden layer as the input to the second network. The size of this hidden layer is flexible and is usually made smaller than the output layer. Hence, this technology is known as the "bottleneck" method [45]. With the bottleneck approach, we successfully trained a hierarchical system with context-dependent state targets [25].

Another possible way of improving hierarchical systems is to downsample the output of the lower network [46]. This expands the time span of the model without increasing the number of input features to the upper net. In our earlier study, we experimentally found the optimal downsampling rate to be 5 [27], and other authors also prefer this value [21, 26].

It is very convenient to train the two networks of the hierarchy in two separate steps, as it requires no modification to the ANN code. However, Veselý showed that better results can be obtained if the two networks are trained as one unit. In his model, during the training of the upper network, the error is propagated back to the lower network as well, so the weights of the lower net are also updated [21]. We successfully applied this solution to DNNs [27], and here, we extend it to convolutional networks as well.

Figure 7 shows the structure of the hierarchical model. Though with the introduction of joint training we no longer have two networks, but just one network with a special structure, for ease of understanding, we will still talk about "upper" and "lower" networks. The lower network operates on a context of several input frames, which in our case will consist of nine consecutive feature vectors [27]. This lower part of the network is evaluated at each frame position of the input data set. However, the upper network uses just every fifth of the resulting vectors. That is, it operates on five vectors coming from the lower part, which are positioned at the $\{0\text{th}, \pm 5\text{th}, \pm 10\text{th}\}$ frame indices of the original input space (the example of Fig. 7 uses only three frames at $\{0, \pm 2\}$). Both the lower and upper network parts may consist of several layers, though experience shows that the upper part requires fewer and smaller layers [23]. Veselý et al. argued that the hierarchical model can be viewed as a convolutional model that performs convolution along time, as it fulfills

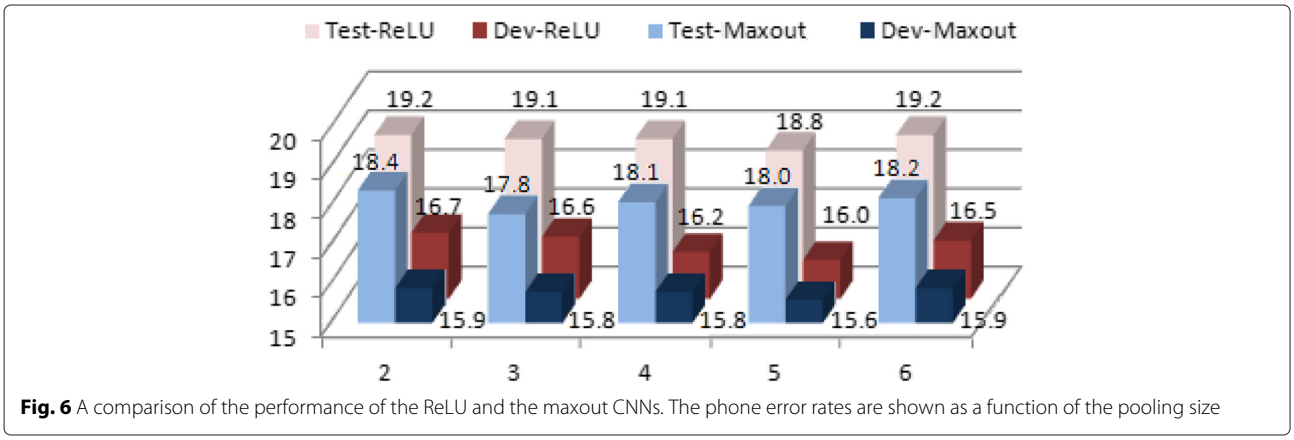

Fig. 6 A comparison of the performance of the ReLU and the maxout CNNs. The phone error rates are shown as a function of the pooling size

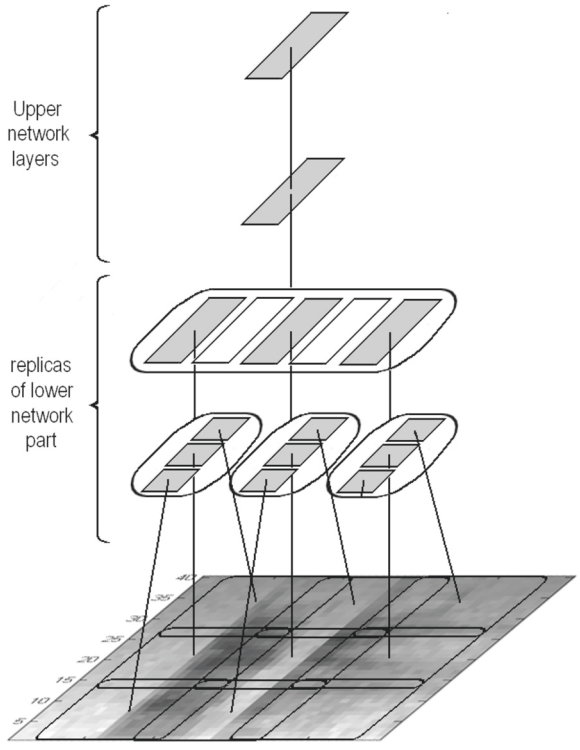

Fig. 7 Schematic diagram of the hierarchical CNN network structure. For clarity, full connections between layers are represented by single lines

the three requirements mentioned in Section 2. Firstly, the lower network part processes only a subset of the input to the whole, joint structure, so the requirement of locality is satisfied. Secondly, it processes several, shifted versions of the local input window using the same sub-network, so weight sharing is also present. Lastly, the output of these local network parts is downsampled before being combined by the higher-level layers. Hence, this model can indeed be called convolutional if we regard downsampling as a special kind of pooling function. Accepting this argumentation, we referred to this sort of architecture as a convolutional model in our earlier paper [27]. However, now we consider this to be slightly misleading, since Veselý's model contains no actual pooling procedure, which is a vital component of all other studies on CNNs—including this one. Thus, in this paper, we prefer to interpret and refer to Veselý's technology as a refined hierarchical model rather than a convolutional model.

The technical details of our implementation of the hierarchical model are as follows. The lower network part consisted of the convolutional network described in earlier sections with two modifications. First, the size of the input layer was decreased from 17 frames to just 9 frames. This input size seems to be sufficient [27] because of the overlap of the local input blocks (see Fig. 7). Thus, the hierarchical model covers 29 frames of input vectors via its 5 local input blocks of 9-9 frames, which overlap by 5 frames. The second modification is that the size of the uppermost hidden layer was reduced to its fifth to form a bottleneck layer. This way, the input to the upper network part (consisting of five vectors from the lower part) was just of the same size as that for all the other layers. The network was the same CNN as that used earlier in all other respects, that is, it consisted of one convolutional layer and three hidden layers of maxout units, with the layer sizes given earlier.

The network was trained by adding one hidden layer after another, using the DPT procedure described earlier. However, after the addition of the bottleneck layer, the training procedure continued by adding the layers of the upper network part. This consisted of two more hidden layers, with 2714 maxout units in each layer. That is, the final hierarchical model contained $4 + 2$ hidden layers.

Table 4 shows that, compared to the four hidden layer CNN, the hierarchical model attains a relative error rate reduction of about 10 % on the development set and 5.5 % on the core test set. One may argue that the comparison is not fair, as the hierarchical model contains two more hidden layers with about 30 % more parameters. Moreover, the input of the CNN consisted of 17 frames of data, while the five local input blocks of the hierarchical model altogether cover 29 frames. The experiments in Section 2.1 revealed that the larger input context cannot explain the improvement in itself (cf. Fig. 3). To prove that the two additional layers cannot explain the better performance either, we trained a six-hidden layer CNN with 29 frames of input. As shown in Table 4, this network performed no better than the four-layer network on the development set and was only slightly better on the test set. This experiment demonstrates that the good performance of the hierarchical model is due to its special structure; that is, the fact that it processes the long context of input in local portions and then combines the results in a hierarchical manner.

5 Dropout

In accordance with the experimental results of other authors, we found that max-pooling CNNs and maxout networks are inclined to overfit the training data set [29, 43]. We already gave an intuitive explanation of this

Table 4 The reduction of the phone error rate using more hidden layers or using the hierarchical modelling scheme

Network type	Development set (%)	Core test set (%)
CNN (4 hidden layers)	15.6	18.0
CNN (6 hidden layers)	15.7	17.7
Hierarchical CNN	14.0	17.0

in Section 3.2. This overfitting behavior was quite strong in the case of the hierarchical model, where we observed a huge gap between the train and development set frame error rates. A technically very simple approach to alleviate overfitting is to use the dropout training technique. During network training, dropout omits each hidden neuron randomly with probability p. This prevents the co-adaptation of neurons, as they cannot directly rely on the activity of other specific neurons [28]. In speech recognition tests, dropout was shown to give significant improvements both in the case of pre-trained sigmoid networks [28] and ReLU networks [36], and now, it is a widely accepted tool in the training of DNNs for speech recognition. Recently, dropout was shown to work nicely in the training of maxout networks as well [18, 19, 29].

Here, we applied dropout during the training of our hierarchical CNN network as follows. First, we used the same dropout rate for each layer, though there exist sophisticated optimization methods for tuning the dropout rate for each layer separately [36]. We varied the dropout rate with a step size of 0.05, and the best result on the development set was obtained with 0.25. We note here that the original paper used a dropout rate of 0.5, but by allowing an extremely large number of training epochs [28]. Most other authors reported the best results with smaller dropout rates between 0.1 and 0.3 [8, 19, 29] perhaps because they used fewer training iterations. For dropout training, we modified our code so that one "epoch" consisted of five sweeps through the data instead of just one. We found that this fivefold increase in the training time was necessary to get good results with dropout.

The effect of dropout training on the performance of the hierarchical maxout CNN is shown in Table 5. The improvement due to dropout is about 5 % on the development set and 3 % on the core test set.

Table 6 compares our result with the best previously reported scores on the TIMIT core test set. Unfortunately, only a few authors used CD phone models like us (indicated by a (CD) remark in the table), so the results are not fully comparable. Still, some main tendencies can be observed. First, all the best results are achieved by special neural structures like CNNs, recurrent nets, and the hierarchical model. Second, systems that use some sophisticated feature set (including speaker adaptation) and context-dependent training targets perform better. We think that the superiority of our solution is due to the CD training targets, the maxout activation function, and the application of the hierarchical structure. Our scores could presumably be improved further by using a refined feature set and/or speaker adaptation.

Table 5 The effect of dropout training on the phone error rates

Network type	Development set (%)	Core test set (%)
Hierarchical maxout CNN	14.0	17.0
Hierarchical maxout CNN + dropout	13.3	16.5

Table 6 The best reported phone error rates (PER) on the TIMIT core test set

Method	PER (%)
Hierachical shallow ANN (CD) [25]	21.2
DNN with softmax units [3]	20.7
DNN with ReLU units [35]	20.8
DNN with ReLU units (CD) [35]	19.8
DNN with dropout [28]	19.7
DNN with BMMI features (CD) [50]	19.1
CNN with speaker adaptation [51]	18.9
DNN + RNN [49]	18.8
CNN with heterogeneous pooling [8]	18.7
LSTM RNN [52]	17.7
CNN with scatter features (CD) [53]	17.4
Hierarchical CNN (CD) [this paper]	16.5

6 Evaluation on a low-resource LVCSR task

The TIMIT corpus is still very useful for quickly testing new ideas in acoustic modelling, as it is a carefully designed corpus with a lot of results available for comparison. However, it is unrealistically small by today's standards, even for research purposes. Hence, we evaluated our best models on a large vocabulary recognition task. The "Szeged" Hungarian Broadcast News Corpus contains 28 h of recordings from eight Hungarian TV channels. Twenty-two hours of data were selected for the training set, 2 h for the development set, and 4 h for the test set. The language model was created using HTK from texts taken from a news portal, and the recognition dictionary consisted of 486,982 words. More details on the corpus and the train/test settings can be found in our earlier paper [47]. We created context-dependent HMM/DNN phone models using a Kullback-Leibler divergence-based state clustering algorithm [48]. This algorithm resulted in 1233 training targets for the neural network. Apart from adjusting the size of the output layer accordingly, all other parameters of the DNNs and CNNs applied were the same as in the TIMIT experiments. As regards convolutional filter size and pooling size, our findings on TIMIT indicated that the CNN is not particularly sensitive to the actual choice of these parameters, so we used the same parameter values that were found to be optimal for TIMIT.

The word error rates attained by the various types of DNN and CNN models are listed in Table 7. As can be seen, the maxout DNN outperformed the ReLU DNN, giving a WER reduction of 2.4 % when trained in one go

Table 7 Word error rates of the DNN and CNN models on the Hungarian LVCSR task

Network type	Development set (%)	Test set (%)
DNN, ReLU	17.7	17.0
DNN, maxout	17.4	16.6
DNN, maxout, hybrid DPT	17.2	16.5
CNN	16.5	15.9
Hierarchical CNN	16.1	15.5

and about 3 % when trained with our "hybrid" discriminative pre-training method. Our results are in accord with the findings of Cai et al., who got an 1–5 % drop in WER using maxout units instead of ReLUs on the Switchboard corpus [17].

Next, we trained a CNN with maxout units, using the pre-training method mentioned above. Compared to the best-performing DNN, the CNN attained a relative WER reduction of about 3.6 %. This result is consistent with the findings of Sainath et al., who reported a 3 % improvement by switching from DNNs to CNNs on a 50-h broadcast news task [14]. Next, we extended the CNN with two more layers to get the hierarchical model described in Section 4. Interestingly, while this model attained a huge drop in the frame error rate (about 17 %), the word the error rate decreased only by 2.5 %. The probable explanation is that the hierarchical scheme can correct the frame errors at positions where the neighboring frames have been recognized correctly. While this is beneficial in pure phone recognition, in word-based recognition, most of these errors can be corrected by the dictionary as well.

7 Conclusions

CNNs seem to be more powerful than fully connected DNNs in cases where the special topology of the input space can be exploited. When applied in speech recognition, CNNs can detect features that are local in frequency, also tolerating small shifts in their positions. Here, we turned the CNN into a hierarchical model, which extends the locality to the time axis as well. We showed that the performance gain provided by this model is indeed due to its special structure and not simply because of the larger input context and the use of more layers. We also experimented with the maxout activation function and showed how it can be readily combined with the pooling function of convolutional neurons. As maxout CNNs outperformed their ReLU counterpart in all the experiments, we plan to use them more frequently in the future. We also think that the p-norm function could yield larger gains, if we could find the best way of using it. Furthermore, the literature suggests that recurrent networks and convolutional networks are currently the two most promising technologies for speech recognition. We are studying the options of combining these two approaches, for example, by the simple methodology employed by Deng et al. [49].

Competing interests
The author declares that he has no competing interests.

Received: 13 December 2014 Accepted: 26 August 2015
Published online: 04 September 2015

References

1. T Chih, P Ru, S Shamma, Multiresolution spectrotemporal analysis of complex sounds. J Acoust Soc Am. **118**, 887–906 (2005)
2. Y Lecun, Y Bengio, in *The Handbook of Brain Theory and Neural Networks*, ed. by MA Arbib. Convolutional networks for images, speech and time series (MIT Press Cambridge, 1995), pp. 255–258
3. A Mohamed, GE Dahl, G Hinton, Acoustic modeling using deep belief networks. IEEE Trans ASLP. **20**(1), 14–22 (2012)
4. GE Dahl, D Yu, L Deng, A Acero, Context-dependent pre-trained deep neural networks for large vocabulary speech recognition. IEEE Trans ASLP. **20**(1), 30–42 (2012)
5. F Seide, G Li, L Chen, D Yu, in *Proc ASRU*. Feature engineering in context-dependent deep neural networks for conversational speech transcription, (2011), pp. 24–29
6. N Jaitly, P Nguyen, A Senior, V Vanhoucke, in *Proc Interspeech*. Application of pretrained deep neural networks to large vocabulary speech recognition, (2012)
7. O Abdel-Hamid, A Mohamed, H Jiang, G Penn, in *Proc ICASSP*. Applying convolutional neural network concepts to hybrid NN-HMM model for speech recognition, (2012), pp. 4277–4280
8. L Deng, O Abdel-Hamid, D Yu, in *Proc ICASSP*. A deep convolutional neural network using heterogeneous pooling for trading acoustic invariance with phonetic confusion, (2013), pp. 6669–6673
9. TN Sainath, A Mohamed, B Kingsbury, B Ramabhadran, in *Proc ICASSP*. Deep convolutional neural networks for LVCSR, (2013), pp. 8614–8618
10. L Tóth, in *Proc ICASSP*. Combining time- and frequency-domain convolution in convolutional neural network-based phone recognition, (2014), pp. 190–194
11. O Abdel-Hamid, L Deng, D Yu, in *Proc Interspeech*. Exploring convolutional neural network structures and optimization techniques for speech recognition, (2013), pp. 3366–3370
12. TN Sainath, B Kingsbury, A Mohamed, G Dahl, G Saon, H Soltau, T Beran, A Aravkin, B Ramabhadran, in *Proc ASRU*. Improvements to deep convolutional neural networks for LVCSR, (2013), pp. 315–320
13. TN Sainath, A Mohamed, B Kingsbury, B Ramabhadran, in *Proc ICASSP*. Joint training of convolutional and non-convolutional neural networks, (2014), pp. 5572–5576
14. TN Sainath, B Kingsbury, G Saon, H Soltau, A Mohamed, G Dahl, B Ramabhadran, Deep convolutional neural networks for large-scale speech tasks. Neural Netw. **64**, 39–48 (2015). doi:10.1016/j.neunet.2014.08.005
15. IJ Goodfellow, D Warde-Farley, M Mirza, A Courville, Y Bengio, in *Proc ICML*. Maxout networks, (2013), pp. 1319–1327
16. X Glorot, A Bordes, Y Bengio, in *Proc AISTATS*. Deep sparse rectifier neural networks, (2011)
17. M Cai, Y Shi, J Liu, in *Proc ASRU*. Deep maxout neural networks for speech recognition, (2013), pp. 291–296
18. Y Miao, F Metze, S Rawat, in *Proc ASRU*. Deep maxout networks for low-resource speech recognition, (2013), pp. 398–403
19. P Swietojanski, J Li, JT Huang, in *Proc ICASSP*. Investigation of maxout networks for speech recognition, (2014), pp. 7649–7653
20. X Zhang, J Trmal, D Povey, S Khudanpur, in *Proc ICASSP*. Improving deep neural network acoustic models using generalized maxout networks, (2014), pp. 215–219
21. K Veselý, M Karafiát, F Grézl, in *Proc ASRU*. Convolutive bottleneck network features for LVCSR, (2011), pp. 42–47
22. H Ketabdar, H Bourlard, Enhanced phone posteriors for improving speech recognition systems. IEEE Trans ASLP. **18**(6), 1094–1106 (2010)
23. J Pinto, G. S. V. S Sivaram, M Magimai-Doss, H Hermansky, H Bourlard, Analysis of MLP based hierarchical phoneme posterior probability estimator. IEEE Trans ASLP. **19**(2), 225–241 (2010)

24. D Vasquez, R Gruhn, W Minker, *Hierarchical neural network structures for phoneme recognition*. (Springer, Berlin, 2013)
25. L Tóth, in *Proc ICASSP*. A hierarchical, context-dependent neural network architecture for improved phone recognition, (2011), pp. 5040–5043
26. Y Zhang, E Chuangsuwanich, J Glass, in *Proc Interspeech*. Language ID-based training of multilingual stacked bottleneck features, (2014), pp. 1–5
27. L Tóth, in *Proc Interspeech*. Convolutional deep rectifier neural nets for phone recognition, (2013), pp. 1722–1726
28. GE Hinton, N Srivastava, A Krizhevsky, I Sutskever, R Salakhutdinov, Improving neural networks by preventing co-adaptation of feature detectors. CoRR. **abs/1207.0580** (2012)
29. M Cai, Y Shi, J Liu, in *Proc ICASSP*. Stochastic pooling maxout networks for low-resource speech recognition, (2014), pp. 3266–3270
30. K-F Lee, H-W Hon, Speaker-independent phone recognition using hidden Markov models. IEEE Trans ASSP. **37**(11), 1641–1648 (1989)
31. X Glorot, Y Bengio, in *Proc AISTATS*. Understanding the difficulty of training deep feedforward neural networks, (2010), pp. 249–256
32. B Kingsbury, in *Proc ICASSP*. Lattice-based optimization of sequence classification criteria for neural-network acoustic modeling, (2009), pp. 3761–3764
33. K Veselý, A Ghoshal, L Burget, D Povey, in *Proc. Interspeech*. Sequence-discriminative training of deep neural networks, (2013), pp. 2345–2349
34. H Bourlard, N Morgan, *Connectionist speech recognition—a hybrid approach*. (Kluwer, Boston, 1994)
35. L Tóth, in *Proc ICASSP*. Phone recognition with deep sparse rectifier neural networks, (2013), pp. 6985–6989
36. GE Dahl, TN Sainath, GE Hinton, in *Proc ICASSP*. Improving deep neural networks for LVCSR using rectified linear units and dropout, (2013), pp. 8609–8613
37. MD Zeiler, M Ranzato, R Monga, M Mao, K Yang, QV Le, P Nguyen, A Senior, V Vanhoucke, J Dean, GE Hinton, in *Proc ICASSP*. On rectified linear units for speech processing, (2013), pp. 3517–3521
38. AL Maas, AY Hannun, AY Ng, in *Proc ICML*. Rectifier nonlinearities improve neural network acoustic models, (2013)
39. J-T Huang, J Li, Y Gong, in *Proc ICASSP*. An analysis of convolutional neural networks for speech recognition, (2015), pp. 4989–4993
40. Y Miao, F Metze, in *Proc Interspeech*. Convolutional neural networks for language-universal feature extraction and cross-language hybrid systems, (2014), pp. 800–804
41. M Cai, Y Shi, J Kang, J Liu, T Su, in *Proc ISCSLP*. Convolutional maxout neural networks for low-resource speech recognition, (2014), pp. 133–137
42. S Renals, P Swietojanski, in *Proc HSCMA*. Neural networks for distant speech recognition, (2014)
43. MD Zeiler, R Fergus, Stochastic pooling for regularization of deep convolutional neural networks. CoRR. **abs/1301.3557** (2013)
44. H Hermansky, D Ellis, S Sharma, in *Proc ICASSP*. Tandem connectionist feature extraction for conventional HMM systems, (2000), pp. 1635–1638
45. C Plahl, R Schlüter, H Ney, in *Proc Interspeech*. Hierarchical bottle neck features for LVCSR, (2010), pp. 1197–1200
46. D Vásquez, G Aradilla, R Gruhn, W Minker, in *Proc ASRU*. A hierarchical structure for modeling inter and intra phonetic information for phoneme recognition, (2009), pp. 124–129
47. T Grósz, L Tóth, in *Text, Speech and Dialogue*, ed. by I Habernal, V Matousek. A comparison of deep neural network training methods for large vocabulary speech recognition (Springer, Berlin, 2013), pp. 36–43
48. G Gosztolya, T Grósz, L Tóth, D Imseng, in *Proc ICASSP*. Building context-dependent DNN acoustic models using Kullback-Leibler divergence-based state tying, (2015), pp. 4570–4574
49. L Deng, J Chen, in *Proc ICASSP*. Sequence classification using the high-level features extracted from deep neural networks, (2014), pp. 6844–6848
50. C Plahl, TN Sainath, B Ramabhadran, D Nahamoo, in *Proc ICASSP*. Improved pre-training of deep belief networks using sparse encoding symmetric machines, (2012), pp. 4165–4168
51. O Abdel-Hamid, H Jiang, in *Proc Interspeech*. Rapid and effective speaker adaptation of convolutional neural network based models for speech recognition, (2013), pp. 1248–1252
52. A Graves, A Mohamed, GE Hinton, in *Proc ICASSP*. Speech recognition with deep recurrent neural networks, (2013), pp. 6645–6649
53. V Peddinti, TN Sainath, S Maymon, B Ramabhadran, D Nahamoo, V Goel, in *Proc ICASSP*. Deep scattering spectrum with deep neural networks, (2014), pp. 210–214

Submit your manuscript to a SpringerOpen journal and benefit from:

- ► Convenient online submission
- ► Rigorous peer review
- ► Immediate publication on acceptance
- ► Open access: articles freely available online
- ► High visibility within the field
- ► Retaining the copyright to your article

Submit your next manuscript at ► springeropen.com